Dynamic Data Structures

Ken Getz
keng@mcwtech.com
@kengetz

Why Dynamic Data Structures?

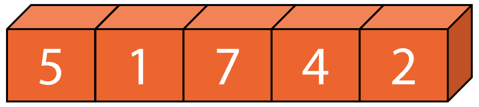
- **VBA provides two data structures**
  - Arrays and collections
- **Each has good and bad features, and compelling reasons to use them**
- **Previously programmed in some other language?**
- **Taken a programming course that covered data structures?**
- **May have encountered dynamic data structures**
  - Like linked lists, stacks, queues, binary trees
- **Can be implemented using arrays or collections**
  - Many reasons not to do that
- **Easiest to use VBA classes**

Three Important Dynamic Data Structures

Stacks | Queues | Ordered Linked Lists

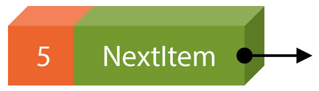
Dynamic vs. Static Data Structures

- **Array is static: If you can predict total number of elements, works fine**
  - Resizing possible, but not efficient
- **Problems?**
  - Arrays are linear
    - Cannot overlay any kind of relationships between elements
  - Arrays are essentially fixed size
    - No way to resize: ReDim simply creates a new array and copies over
  - Arrays use space inefficiently
    - Declare an array to hold 50 elements and use 5? Wasting extra space
- **Dynamic data structure grows or shrinks as necessary**
  - Allocate new storage when needed; discard it when done



Dynamic Data Structures

- **So what is a dynamic data structure?**
- **Generally consists of:**
  - Simple data storage (can be as complex as you like)
  - At least one link to another instance of the same class
    - Called a "pointer" or a "reference"



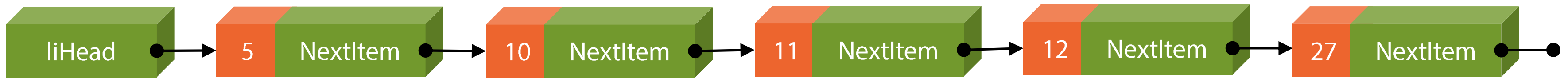
```
Class StackItem
   Dim Value As Variant
   Dim NextItem As StackItem
End Class
```

It's a Big Topic

- **General discussion of data structures normally covered in a full college-level course**
- **Only learn the basics here**
  - Enough to get started

Linear Data Structures

- **Simplest dynamic data structures (and all the ones covered here) are linear**
  - Each element contains information and a reference to another element of the same type
  - Easy to add and remove elements in any position
  - Easy to resize—simply insert a new element or delete an element
- **Generally includes at least one header element**
  - A reference to the type contained in the list



Differentiating Linear Data Structures

- **What differentiates?**
- **Arbitrary rules about how you add or delete nodes**
  - Stacks and queues are both linear linked lists
  - Stack only accepts new items at its "top"
  - Stack removes items from the same place
  - Queue only accepts new items at its "rear"
  - Queue removes items from its "front"
- **Linked list can have links in both directions**
  - So you can traverse the list in either order
- **Next step?**
  - Data structures with two links, like binary trees

In Memory Only!

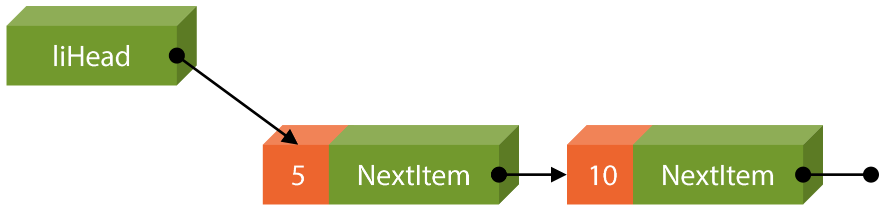
- **Term "Dynamic Data Structures" always refers to in-memory data structures**
- **Techniques covered here deal only with work in current instance of your application**
- **Need to store on disk?**
  - Need some means of serializing the data; a way to store and retrieve the data in permanent storage
  - Use VBA's disk file handling to manage storage
- **Use these data structures once you have retrieved the data from disk**

Using VBA to Model Dynamic Data Structures

- **Use class to represent each element of the list**
- **Instantiate new instance when needed**
  - Fill in data
  - Set link to next instance of the class in the list
- **Generally, need two classes**
  - One for data structure
  - One for each element of the structure
- **For stack:**
  - One class contains pointer to top of the stack (Stack class)
  - One class contains data and reference to next element (StackItem class)

Working with References

- **See a reference name?**
  - Think "The thing that <reference name> points to"



- **Assume that ListItem class contains:**
  - Value (Variant)
  - NextItem (ListItem)
- **liHead is of type ListItem**
- **In code, see "liHead", think "ListItem that liHead points to"**
  - liHead.Value is 5
  - liHead.NextItem.Value is 10
- **Uninitialized pointer refers to Nothing**
  - Represented by end "dot" in diagrams here

Comparing References

- **Use equal sign to compare values, not references**

```
If x = 5 Then
End If
```

- **Use Is operator to compare object references**

```
If liHead Is Nothing Then
  ' You know liHead is uninitialized
End If
```

- **Can use Is to compare two references**
  - Do the references point to the same objects?

```
If liHead Is liNew Then
  ' You know they point to the same object
End If
```

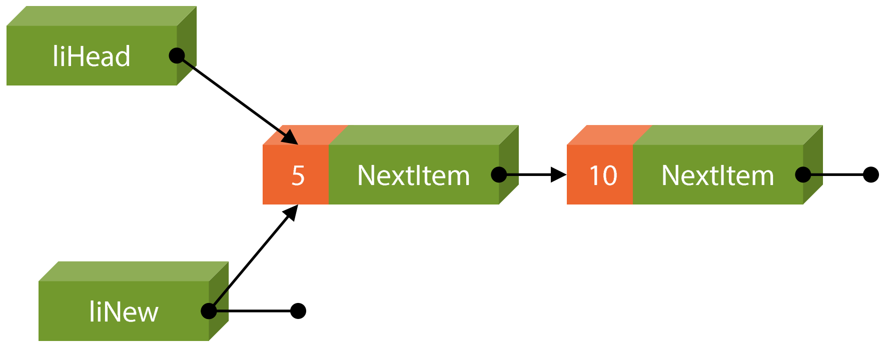
- **How to check if reference isn't Nothing?**

```
If Not liHead Is Nothing Then
  ' You know liHead points to something
End If
```

Working with References

- **Often need to make an object refer to existing item**
- **Just as when using Set to point a reference to a new item**
  - Can refer to existing item as well

```
Dim liNew As ListItem
Set liNew = liHead
```

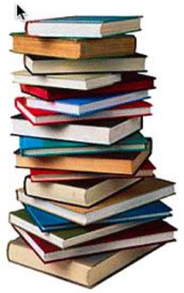


- **Set reference to Nothing to break a link**

```
Set liNew = Nothing
```

- **If no other reference to object, VBA destroys it**

Stack Data Structure



- **Term "Stack" comes from every-day usage**
  - Stack of dishes, stack of books, stack of chairs
- **Unique behavior:**
  - Last item added is first removed: **L**ast **I**n **F**irst **O**ut (**LIFO**)
- **Important property**
  - **StackEmpty**: Returns True if stack is empty

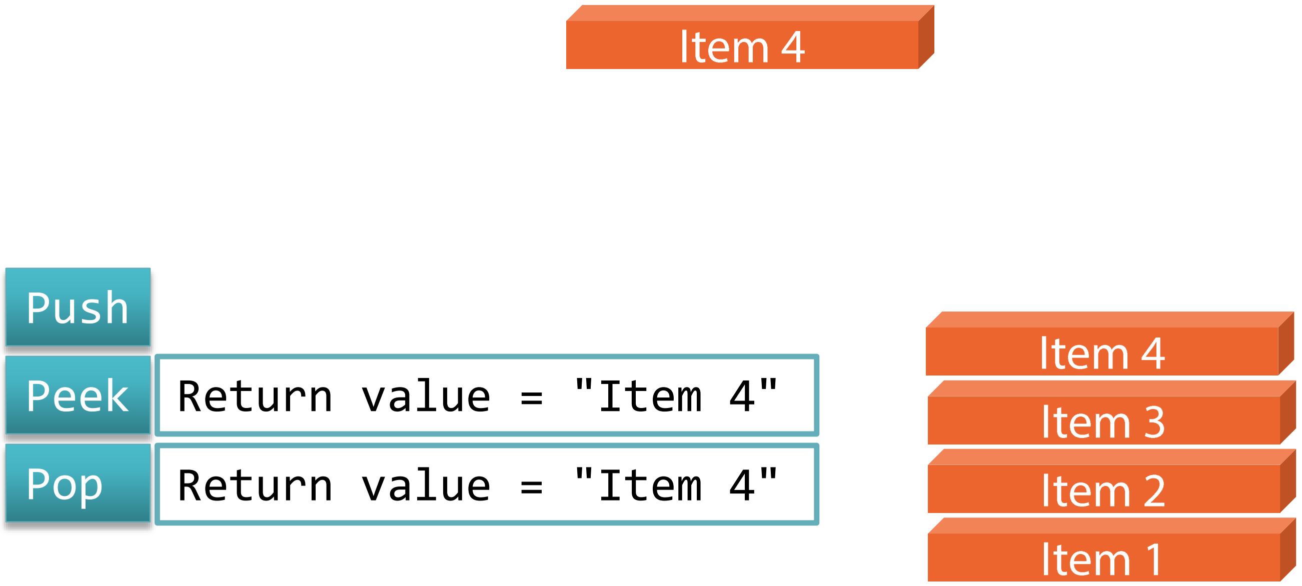
Operations on a Stack

Push an item on the top

Pop an item from the top

Peek to retrieve value of top item

Working with a Stack



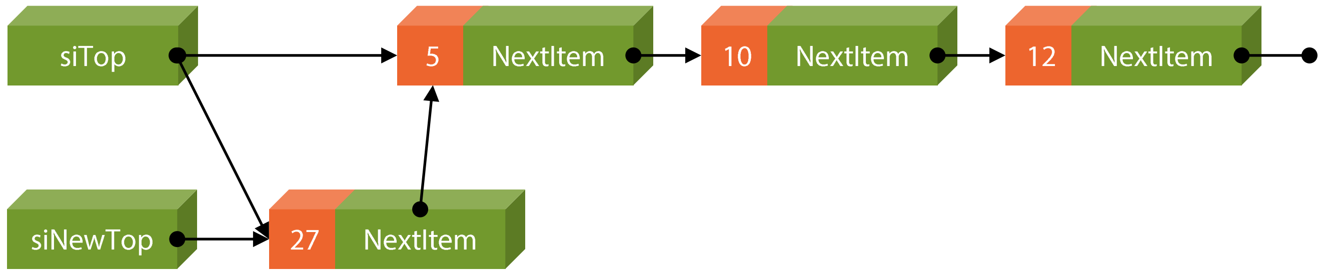
```
Return value = "Item 4"
```

```
Return value = "Item 4"
```

DEMO

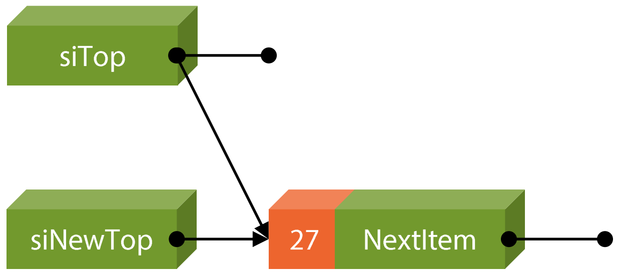
- View the Stack and StackItem code

Push an Item On a Stack



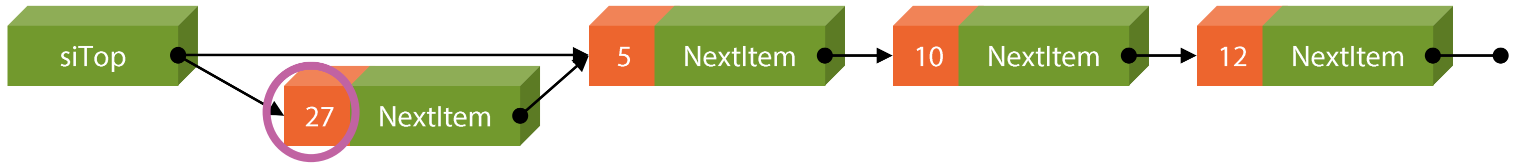
```
Dim siNewTop as StackItem
Set siNewTop = New StackItem
siNewTop.Value = varText
Set siNewTop.NextItem = siTop
Set siTop = siNewTop
```

Starting with an Empty Stack



```
Dim siNewTop as StackItem
Set siNewTop = New StackItem
siNewTop.Value = varText
Set siNewTop.NextItem = siTop
Set siTop = siNewTop
```

Pop an Item From a Stack



```
Pop = siTop.Value
Set siTop = siTop.NextItem
```

DEMO

- Run StackTests demonstration

Introducing the Queue

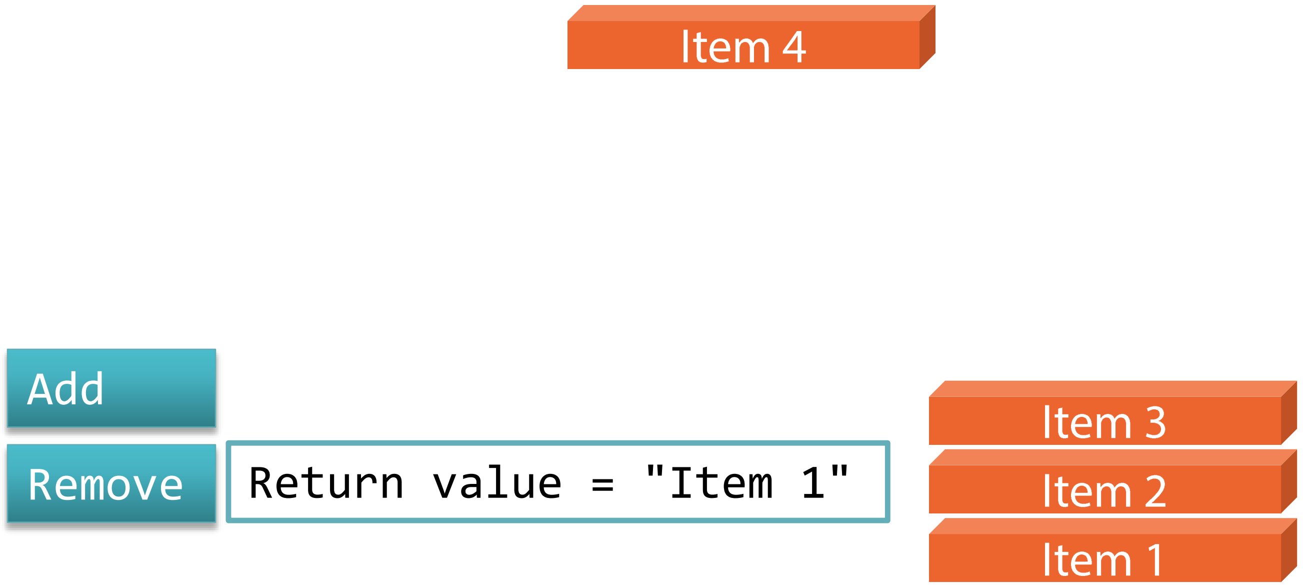
- **Queue** like a line of people
- First person to join is first person to be served,
  or leave the line
  - New people added to queue at back, or rear
  - People leave queue from the front
- **Unique behavior:**
  - First item added is first removed: **F**irst **I**n **F**irst **O**ut (**FIFO**)
- **IsEmpty property returns true if queue is empty**
  - Both front and rear pointers are Nothing

Operations on a Queue

Add a new Item at the rear

Remove an item from the front

Working with a Queue

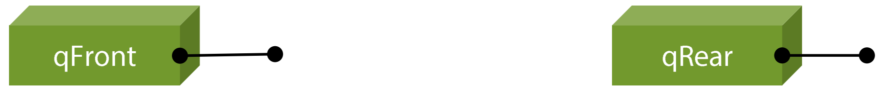


```
Return value = "Item 1"
```

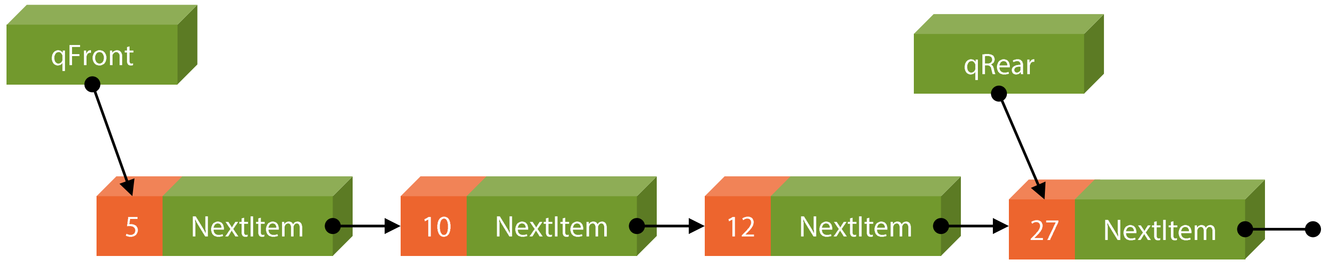
DEMO

- Investigate Queue and QueueItem classes

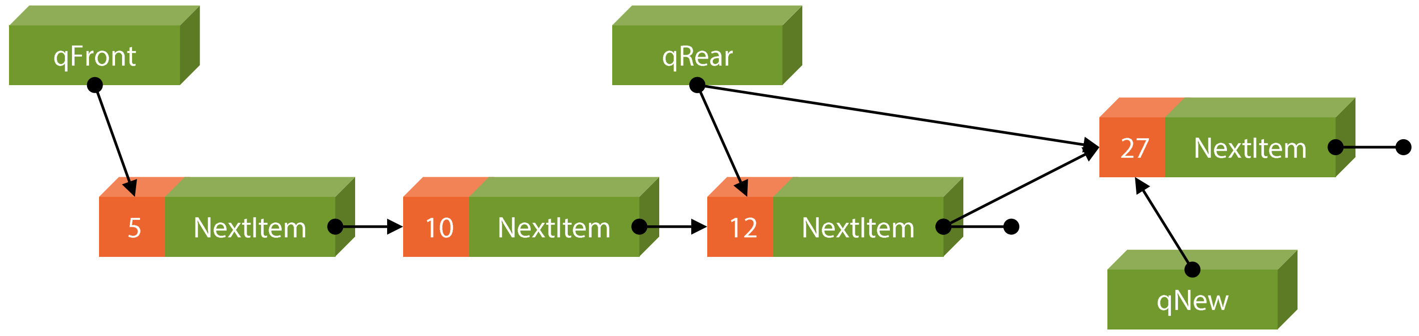
How to Tell if a Queue is Empty



```
Public Property Get IsEmpty() As Boolean
    IsEmpty = ((qFront Is Nothing) And (qRear Is Nothing))
End Property
```

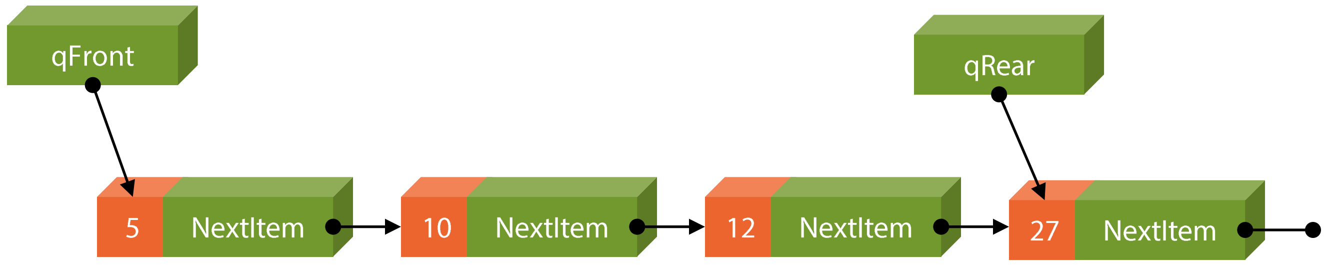


Adding an Item to a Queue



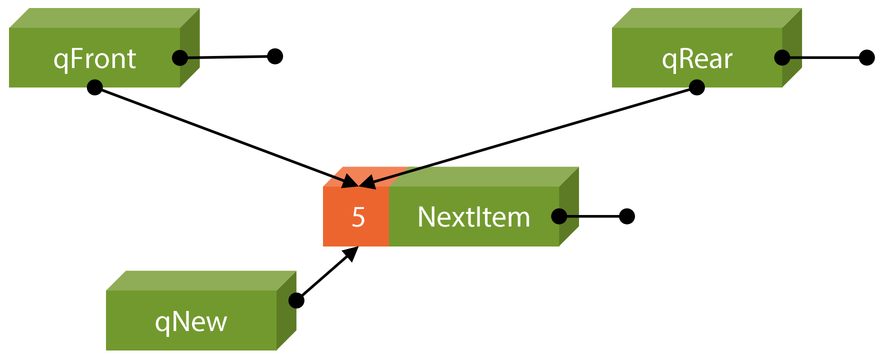
```
Dim qNew As QueueItem
Set qNew = New QueueItem
qNew.Value = varNewItem
If IsEmpty Then
    Set qFront = qNew
    Set qRear = qNew
Else
    Set qRear.NextItem = qNew
    Set qRear = qNew
End If
```

Adding an Item to a Queue



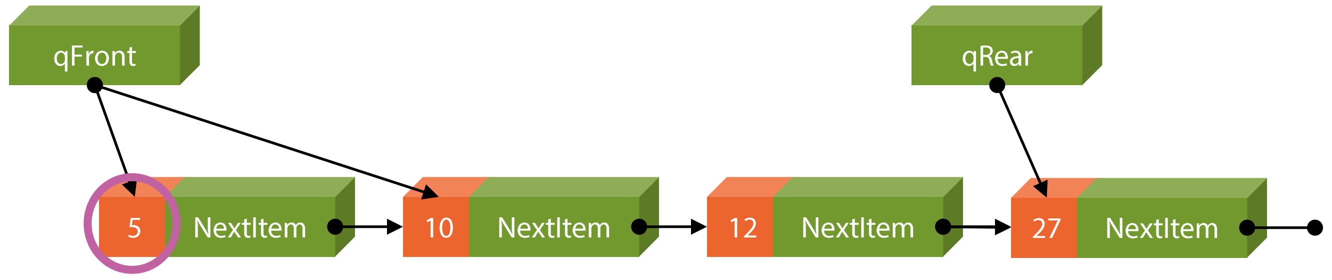
```
Dim qNew As QueueItem
Set qNew = New QueueItem
qNew.Value = varNewItem
If IsEmpty Then
    Set qFront = qNew
    Set qRear = qNew
Else
    Set qRear.NextItem = qNew
    Set qRear = qNew
End If
```

Adding an Item to an Empty Queue



```
Dim qNew As QueueItem
Set qNew = New QueueItem
qNew.Value = varNewItem
If IsEmpty Then
    Set qFront = qNew
    Set qRear = qNew
Else
    Set qRear.NextItem = qNew
    Set qRear = qNew
End If
```

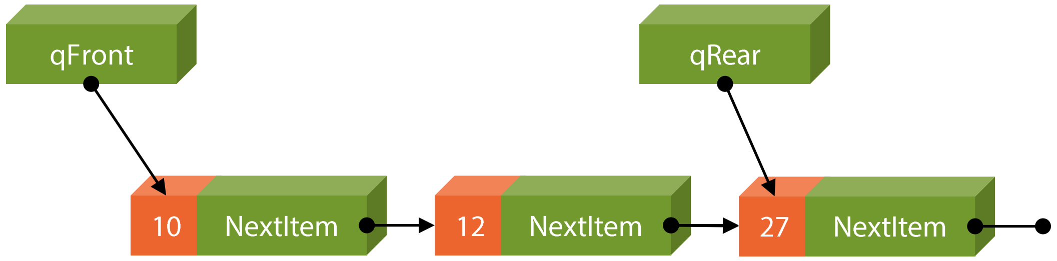
Removing an Item from a Queue



```
If IsEmpty Then
    Remove = Null
Else
    Remove = qFront.Value
    If qFront Is qRear Then
        Set qFront = Nothing
        Set qRear = Nothing
    Else
        Set qFront = qFront.NextItem
    End If
End If
```

Return value = 5

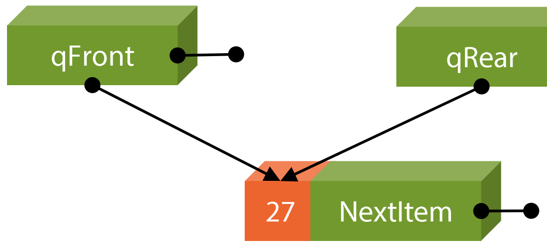
Removing an Item from a Queue



```
If IsEmpty Then
    Remove = Null
Else
    Remove = qFront.Value
    If qFront Is qRear Then
        Set qFront = Nothing
        Set qRear = Nothing
    Else
        Set qFront = qFront.NextItem
    End If
End If
```

```
Return value = 5
```

Removing the Last Item from a Queue



```
If IsEmpty Then
    Remove = Null
Else
    Remove = qFront.Value
    If qFront Is qRear Then
        Set qFront = Nothing
        Set qRear = Nothing
    Else
        Set qFront = qFront.NextItem
    End If
End If
```

```
Return value = 27
```

DEMO

- Run Queue demo

Linked List

HEATHER IN SHOCK: Richie Dates Denise
SPECIAL DOUBLE ISSUE
People
Exclusive Photos!
Brad & Angelina In Africa
100 MOST BEAUTIFUL
ANGELINA JOLIE
PLUS 64 Pages Of Other Stunners
CARNIE REVEALS Her Battles With Food & Alcohol

- **List is collection of items with an implied order**
  - Order may not be significant, but all linked lists have an order
- **Stacks and Queues are special implementations**
  - Rather than restricting access, Linked List provides access to all elements
- **Special characteristics:**
  - Items in the list are normally in incremental order

Operations on a Linked List

Add a New Item

Delete an Existing Item

Traverse the List

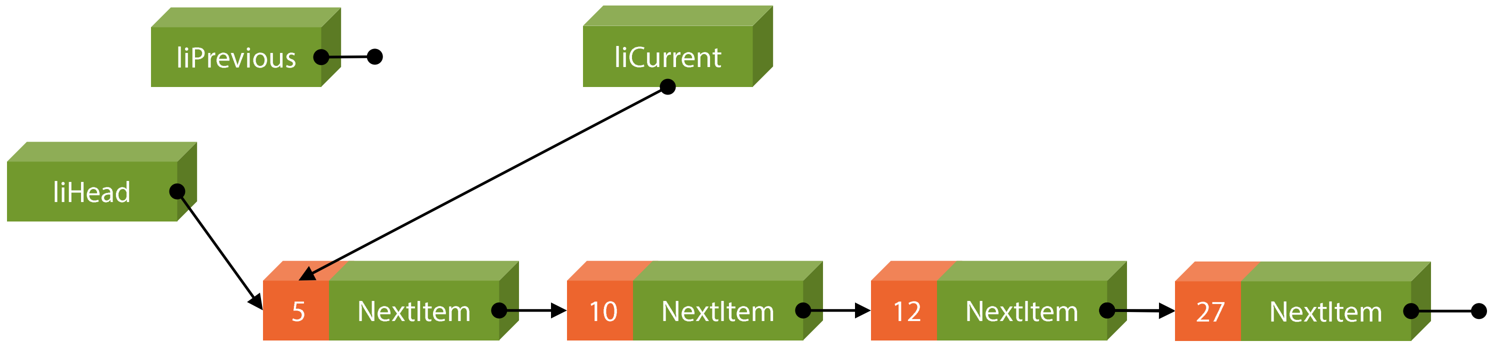
Working with a Linked List

- **Both Add and Delete methods count on private method, Search, to locate item in the list**
- **Search method accepts three parameters**
  - **Value** to locate (Variant)
  - **Current list item** (ListItem, passed ByRef): The matching item
  - **Previous list item** (ListItem, passed ByRef): The previous item in list
    - Previous list item will be Nothing if Current is first item
- **Search method returns Boolean indicating whether it found a match in the list**
  - Also sets liCurrent and liPrevious, so other code can insert or delete items

DEMO

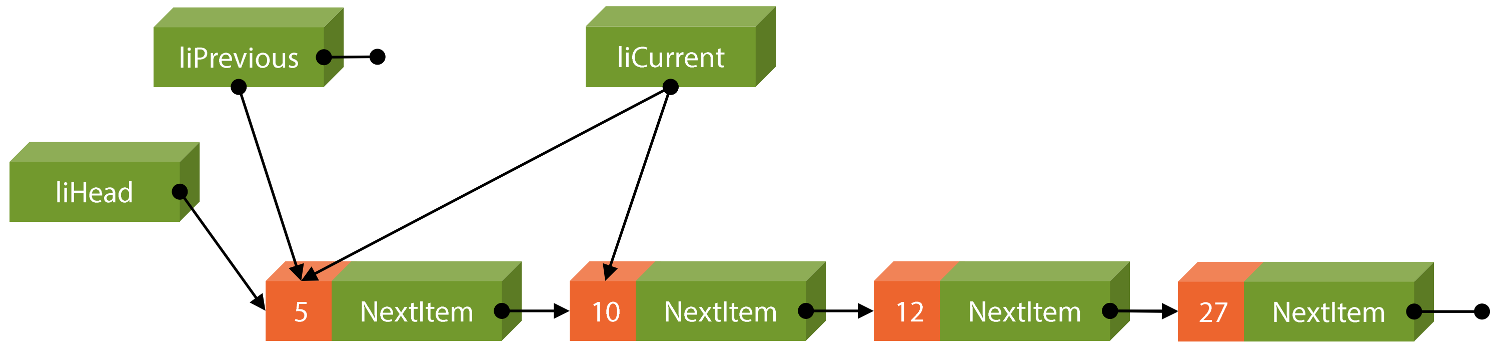
- Investigate List and ListItem classes

Searching In a List: Find a Place for 20



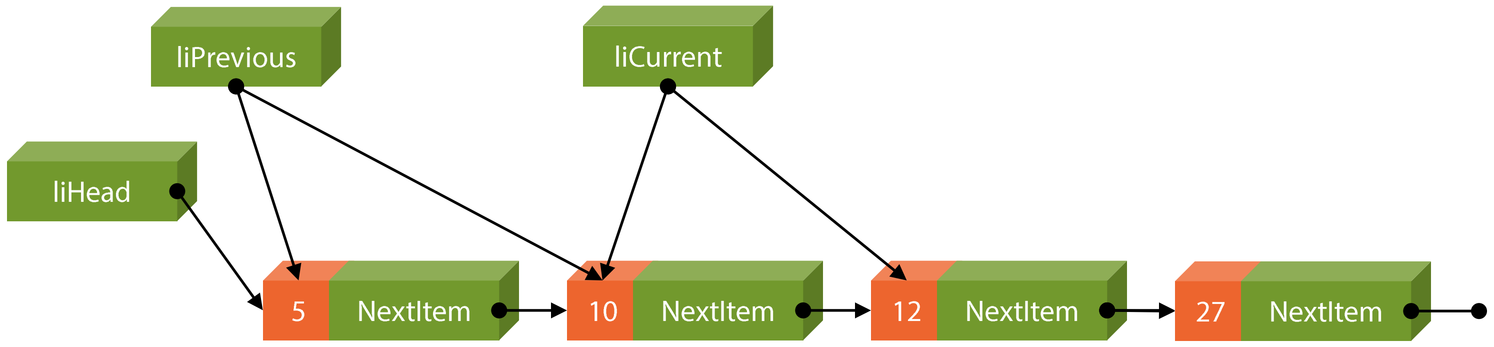
```
' In the Search method
Dim blnFound As Boolean
blnFound = False

Set liPrevious = Nothing
Set liCurrent = liHead

Do While Not liCurrent Is Nothing
  ' Look for a value greater than or equal to 20
Loop
```

Searching In a List: Find a Place for 20



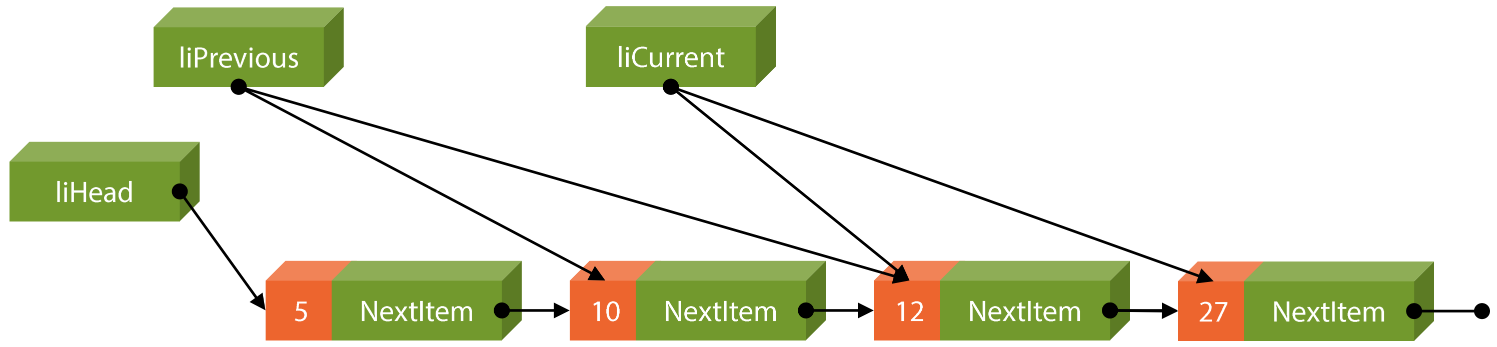
```
' Inside loop
If varItem > liCurrent.Value Then
    Set liPrevious = liCurrent
    Set liCurrent = liCurrent.NextItem
Else
    Exit Do
End If
```

Searching In a List: Find a Place for 20



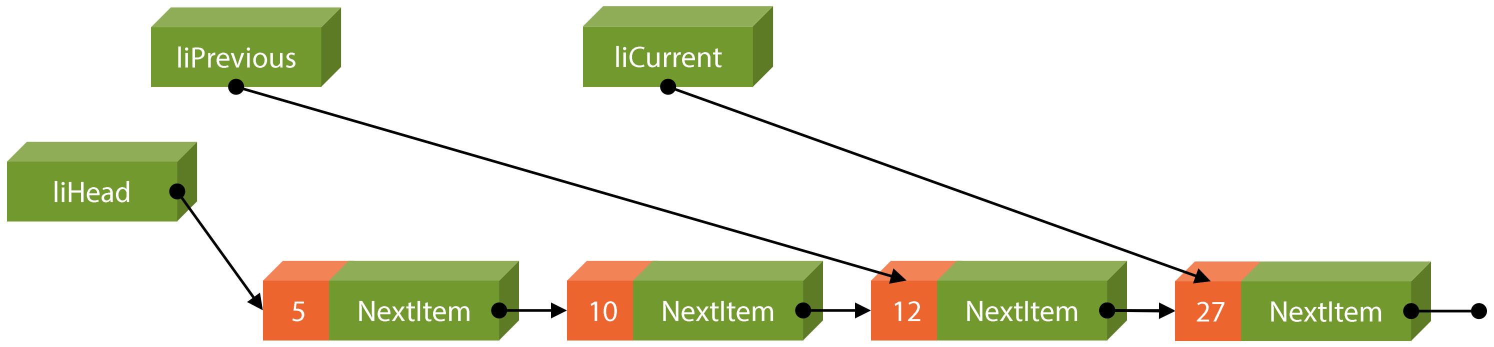
```
' Inside loop
If varItem > liCurrent.Value Then
    Set liPrevious = liCurrent
    Set liCurrent = liCurrent.NextItem
Else
    Exit Do
End If
```

Searching In a List: Find a Place for 20



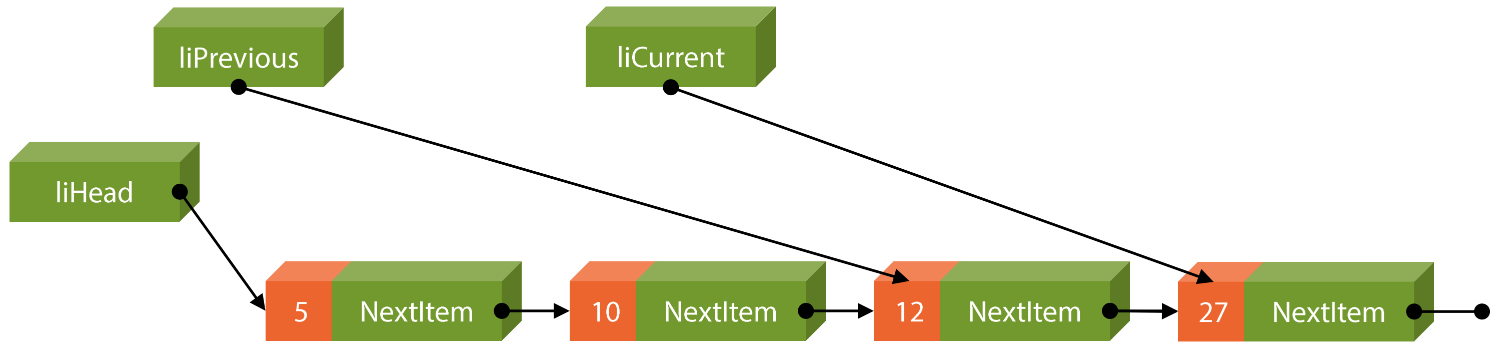
```
' Inside loop
If varItem > liCurrent.Value Then
    Set liPrevious = liCurrent
    Set liCurrent = liCurrent.NextItem
Else
    Exit Do
End If
```

Searching In a List: Find a Place for 20



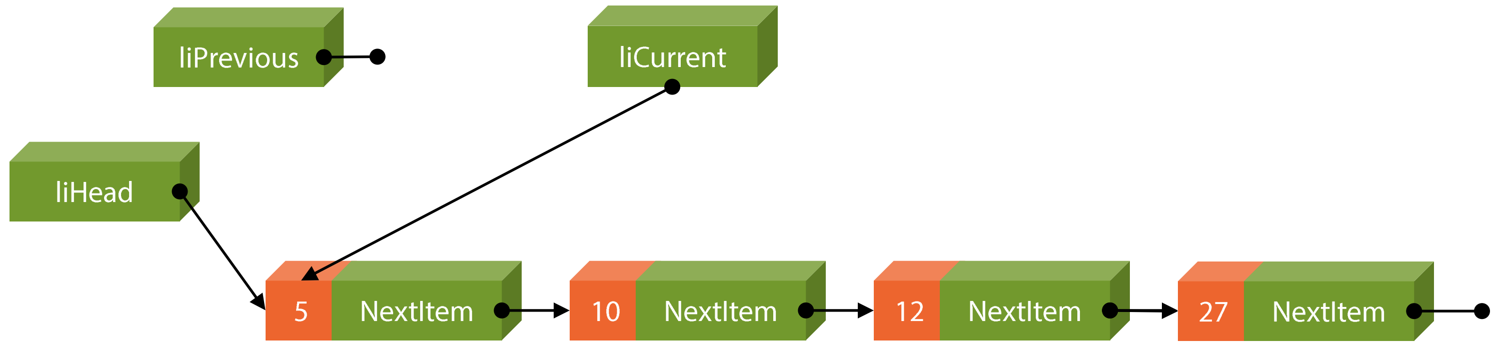
```
' Inside loop
If varItem > liCurrent.Value Then
    Set liPrevious = liCurrent
    Set liCurrent = liCurrent.NextItem
Else
    Exit Do
End If
```

Searching In a List: Find a Place for 20



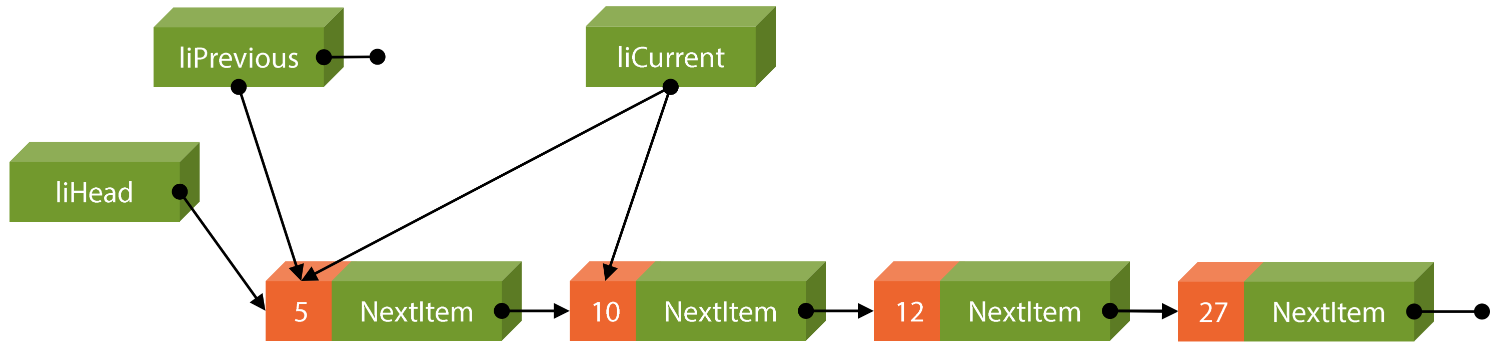
```
' After loop
If Not liCurrent Is Nothing Then
    blnFound = (liCurrent.Value = varItem)
End If
' Set the return value (False, in this case)
Search = blnFound
```

Searching In a List: Find a Place for 12



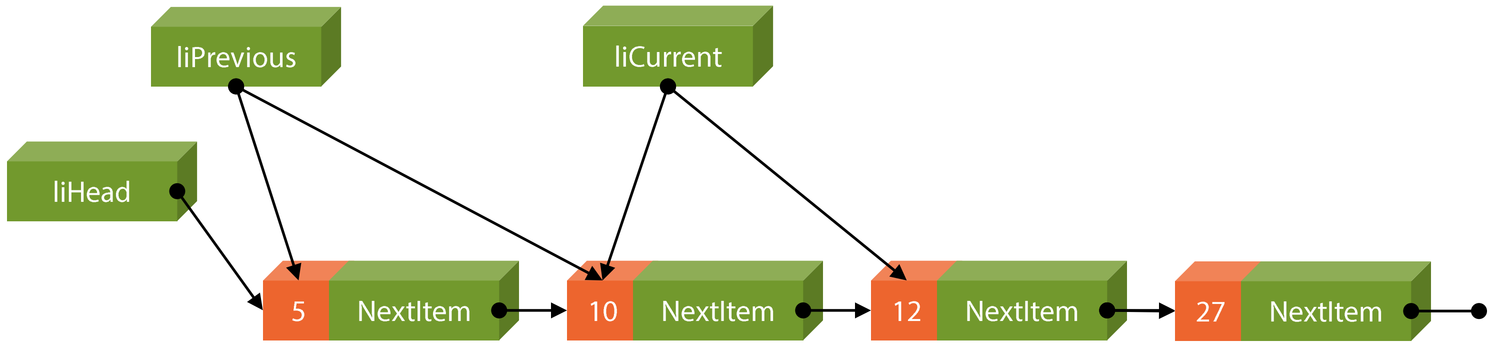
```
' In the Search method
Dim blnFound As Boolean
blnFound = False

Set liPrevious = Nothing
Set liCurrent = liHead

Do While Not liCurrent Is Nothing
  ' Look for a value greater than or equal to 12
Loop
```

Searching In a List: Find a Place for 12



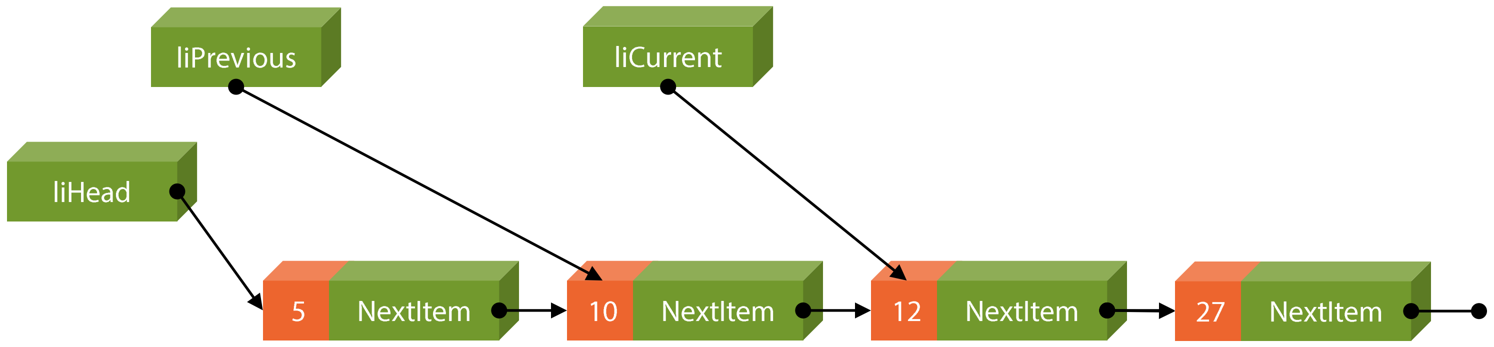
```
' Inside loop
If varItem > liCurrent.Value Then
    Set liPrevious = liCurrent
    Set liCurrent = liCurrent.NextItem
Else
    Exit Do
End If
```

Searching In a List: Find a Place for 12



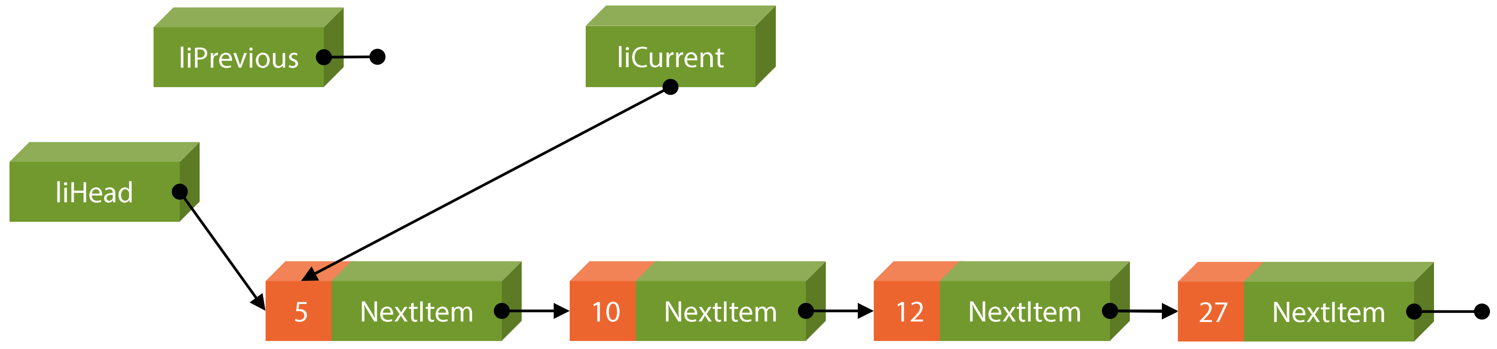
```
' Inside loop
If varItem > liCurrent.Value Then
    Set liPrevious = liCurrent
    Set liCurrent = liCurrent.NextItem
Else
    Exit Do
End If
```

Searching In a List: Find a Place for 12



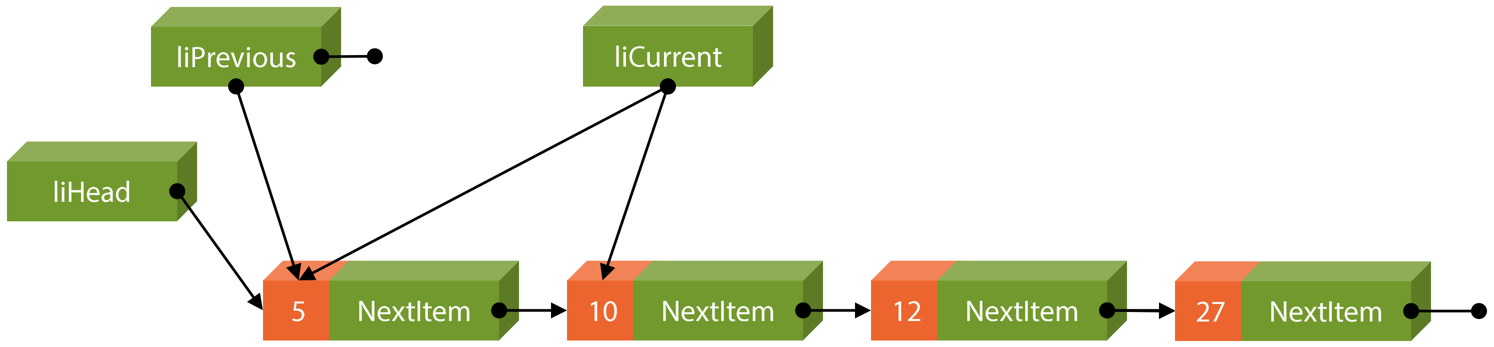
```
' After loop
If Not liCurrent Is Nothing Then
    blnFound = (liCurrent.Value = varItem)
End If
' Set the return value (True, in this case)
Search = blnFound
```

Searching In a List: Find a Place for 54



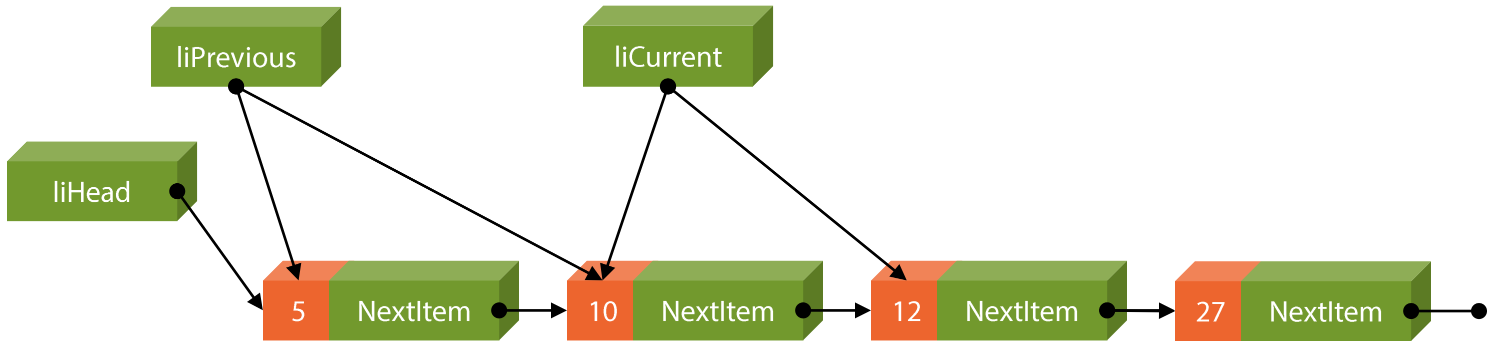
```
' In the Search method
Dim blnFound As Boolean
blnFound = False

Set liPrevious = Nothing
Set liCurrent = liHead

Do While Not liCurrent Is Nothing
  ' Look for a value greater than or equal to 54
Loop
```

Searching In a List: Find a Place for 54



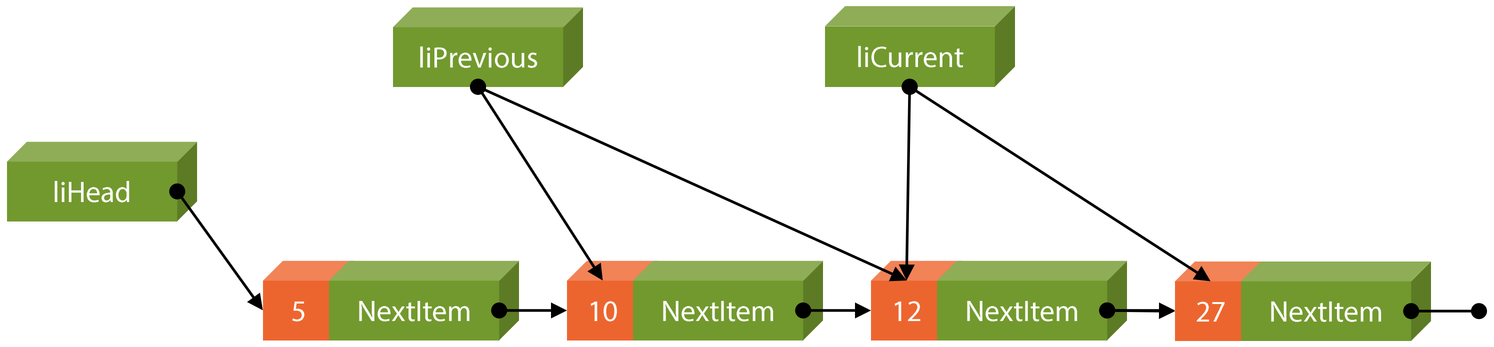
```
' Inside loop
If varItem > liCurrent.Value Then
    Set liPrevious = liCurrent
    Set liCurrent = liCurrent.NextItem
Else
    Exit Do
End If
```

Searching In a List: Find a Place for 54



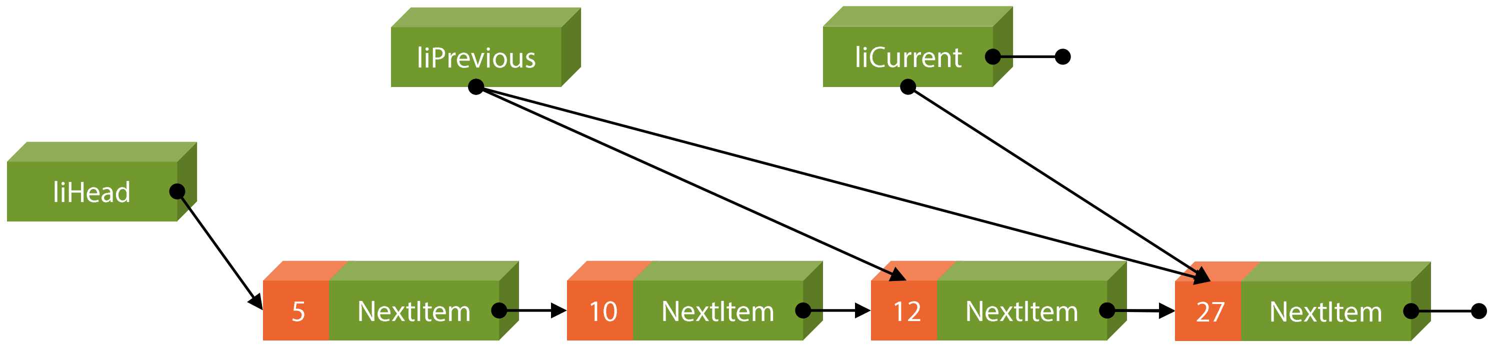
```
' Inside loop
If varItem > liCurrent.Value Then
    Set liPrevious = liCurrent
    Set liCurrent = liCurrent.NextItem
Else
    Exit Do
End If
```

Searching In a List: Find a Place for 54



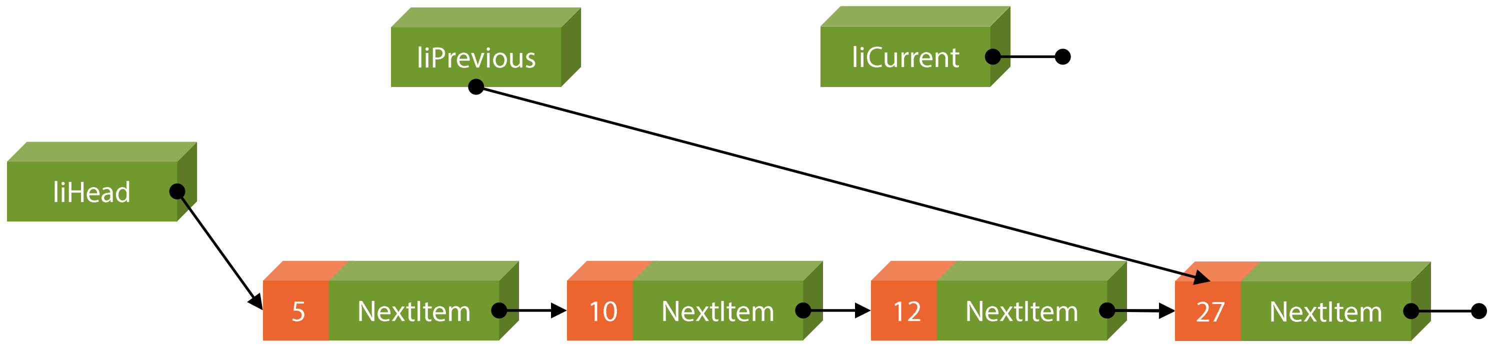
```
' Inside loop
If varItem > liCurrent.Value Then
    Set liPrevious = liCurrent
    Set liCurrent = liCurrent.NextItem
Else
    Exit Do
End If
```

Searching In a List: Find a Place for 54



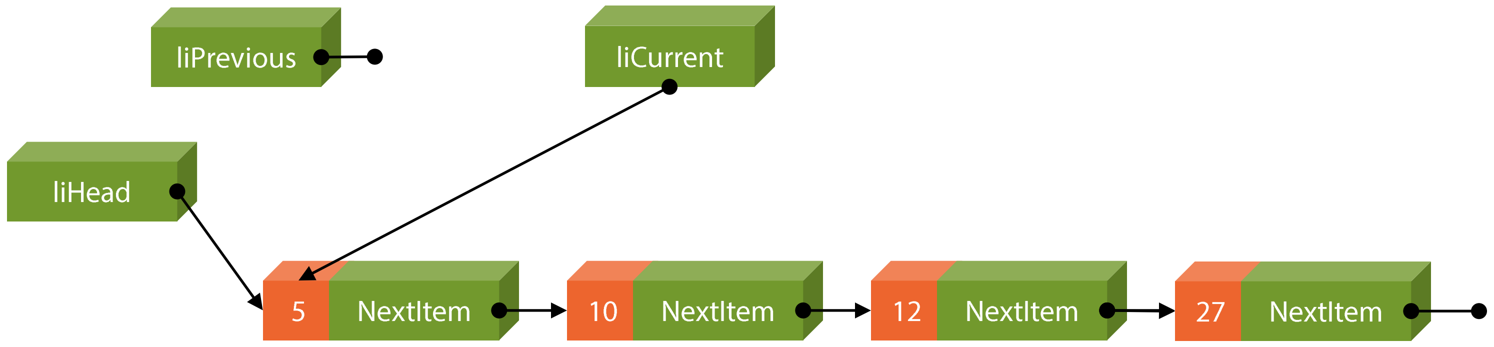
```
' Inside loop
If varItem > liCurrent.Value Then
    Set liPrevious = liCurrent
    Set liCurrent = liCurrent.NextItem
Else
    Exit Do
End If
' liCurrent Is Nothing: Loop ends
```

Searching In a List: Find a Place for 54



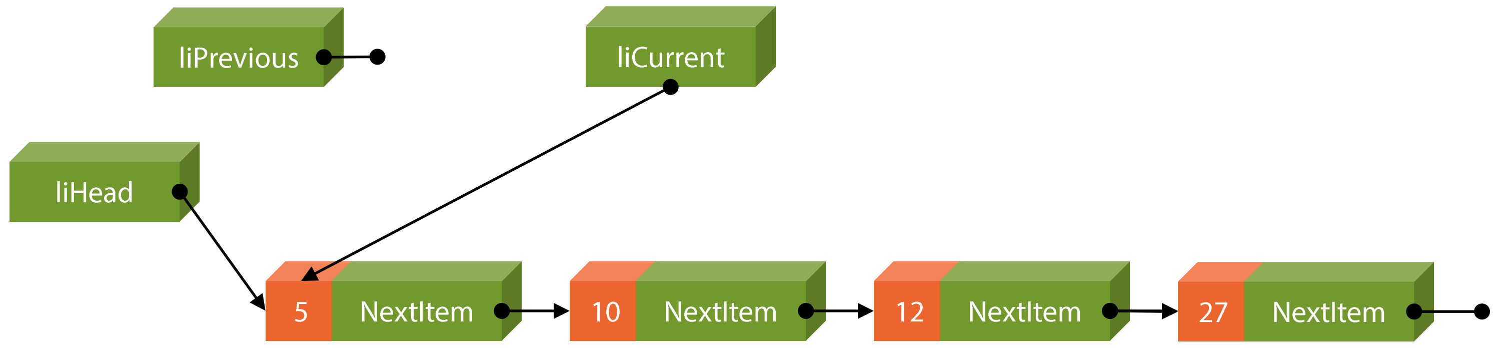
```
' After loop
If Not liCurrent Is Nothing Then
    blnFound = (liCurrent.Value = varItem)
End If
' Set the return value (False, in this case)
Search = blnFound
```

Searching In a List: Find a Place for 2



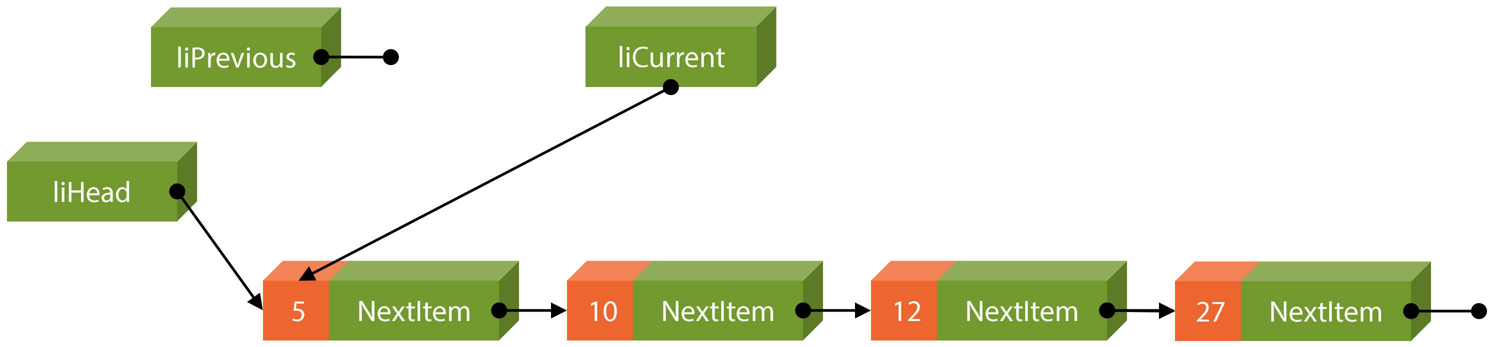
```
' In the Search method
Dim blnFound As Boolean
blnFound = False

Set liPrevious = Nothing
Set liCurrent = liHead

Do While Not liCurrent Is Nothing
  ' Look for a value greater than or equal to 2
Loop
```

Searching In a List: Find a Place for 2



```
' Inside loop
If varItem > liCurrent.Value Then
    Set liPrevious = liCurrent
    Set liCurrent = liCurrent.NextItem
Else
    Exit Do
End If
```

Searching In a List: Find a Place for 2

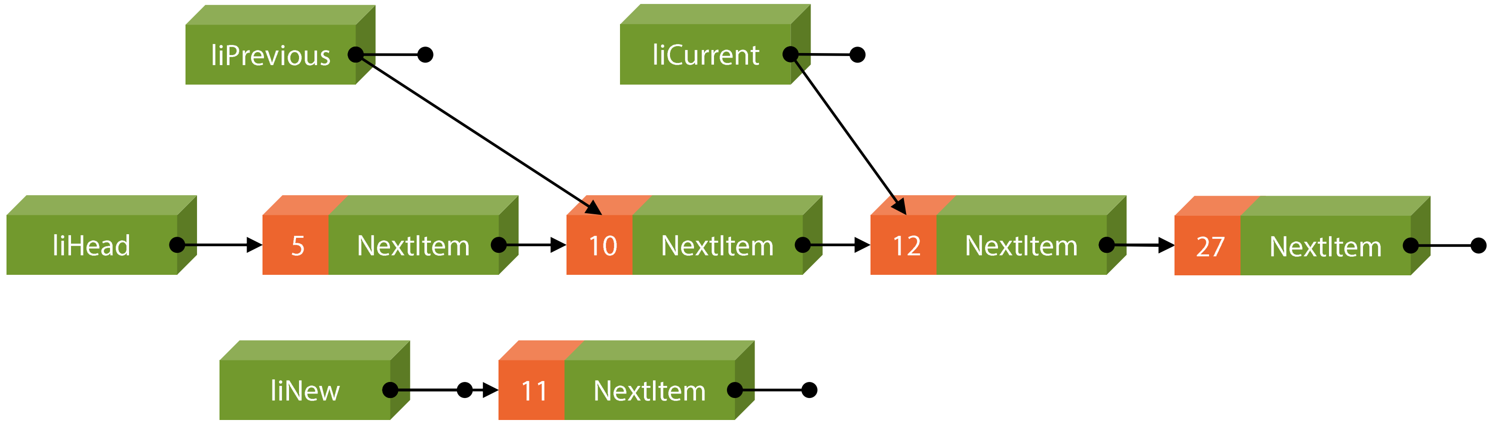


```
' After loop
If Not liCurrent Is Nothing Then
    blnFound = (liCurrent.Value = varItem)
End If
' Set the return value (False, in this case)
Search = blnFound
```

Adding Item to an Ordered List

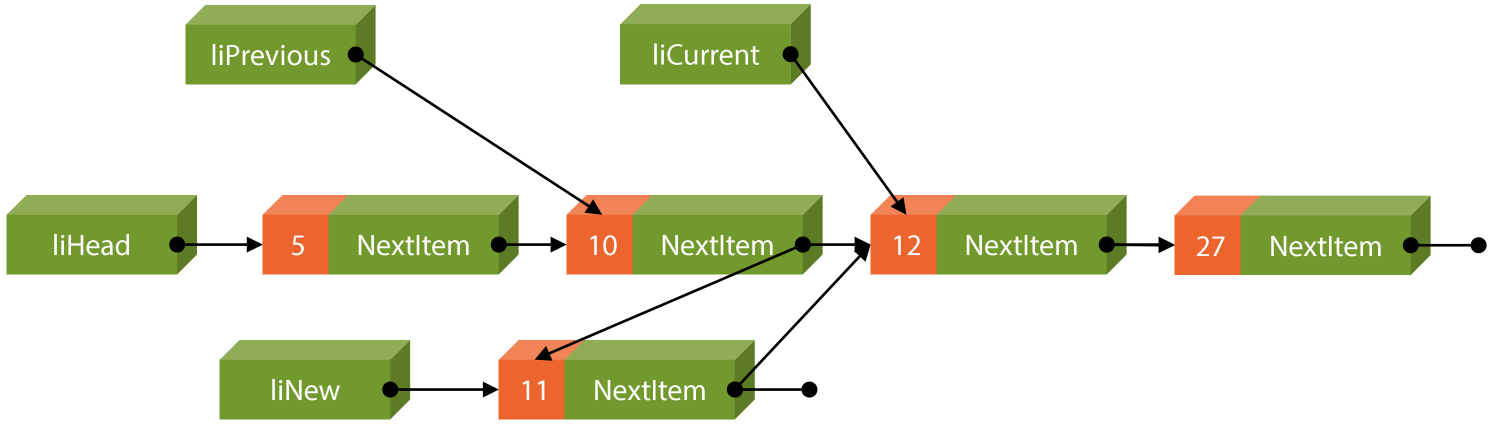
- **Want to keep the list in order, so position significant**
- **Create new ListItem, set value**
- **Call Search method to locate position**
- **Two scenarios:**
  - New position before the beginning of the list
  - New position after the beginning of the list

Adding An Item to an Ordered List (Middle)



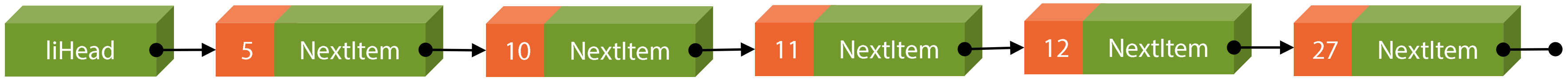
```
' varValue = 11
Set liNew = New ListItem
liNew.Value = varValue

Call Search(varValue, liCurrent, liPrevious)
```

Adding An Item to an Ordered List (Middle)



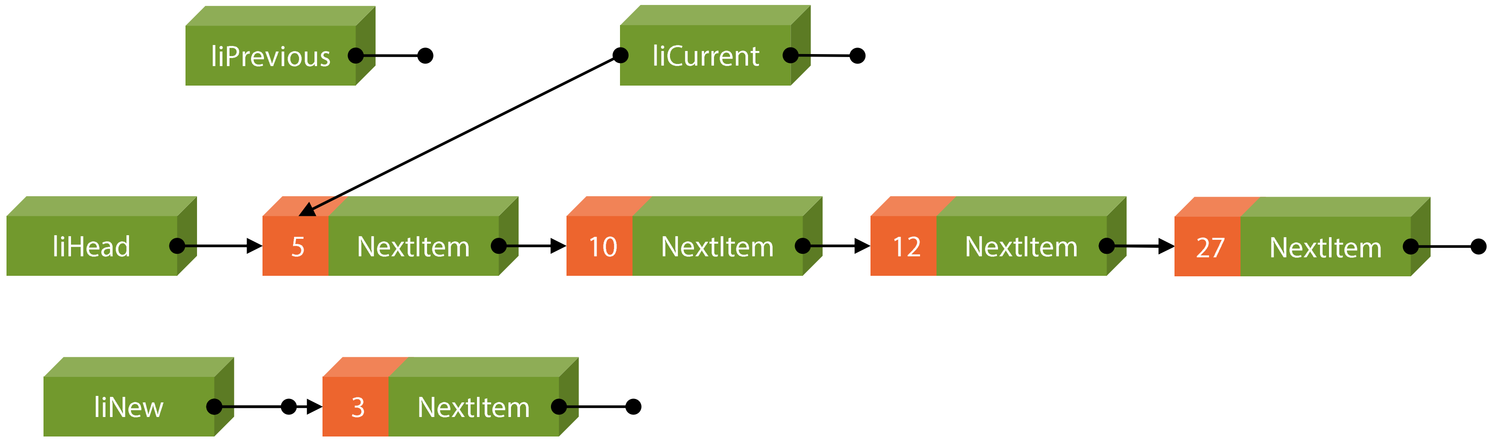
```
If Not liPrevious Is Nothing Then
    Set liNew.NextItem = liPrevious.NextItem
    Set liPrevious.NextItem = liNew
Else
    Set liNew.NextItem = liHead
    Set liHead = liNew
End If
```

Adding An Item to an Ordered List (Middle)



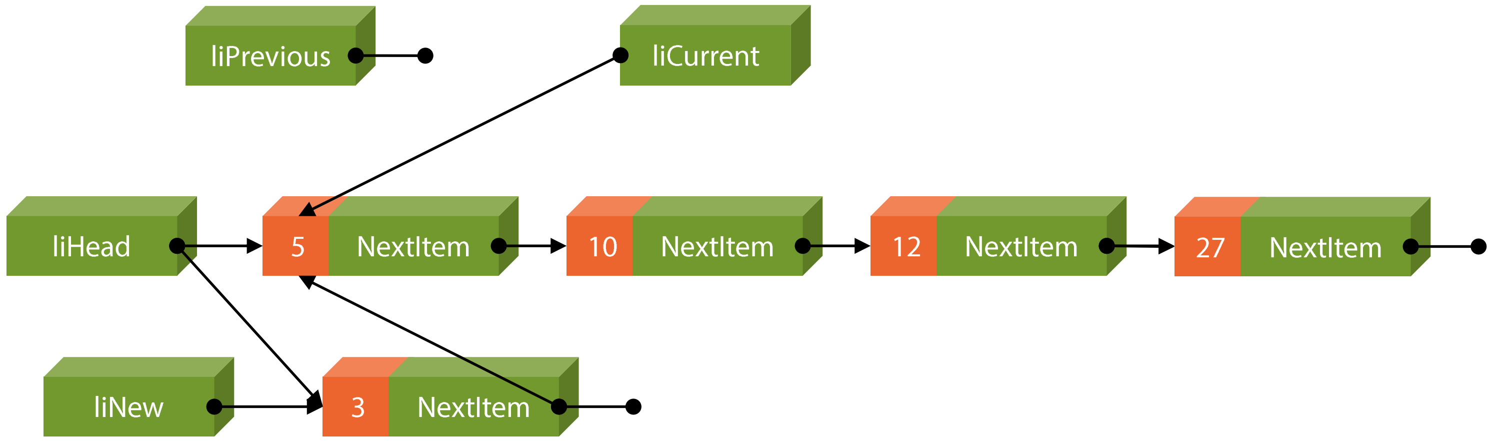
```
If Not liPrevious Is Nothing Then
    Set liNew.NextItem = liPrevious.NextItem
    Set liPrevious.NextItem = liNew
Else
    Set liNew.NextItem = liHead
    Set liHead = liNew
End If
```

Adding An Item to an Ordered List (Beginning)



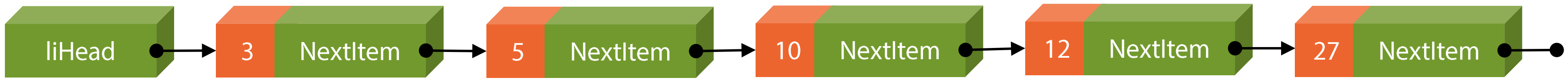
```
' varValue = 3
Set liNew = New ListItem
liNew.Value = varValue

Call Search(varValue, liCurrent, liPrevious)
```

Adding An Item to an Ordered List (Beginning)



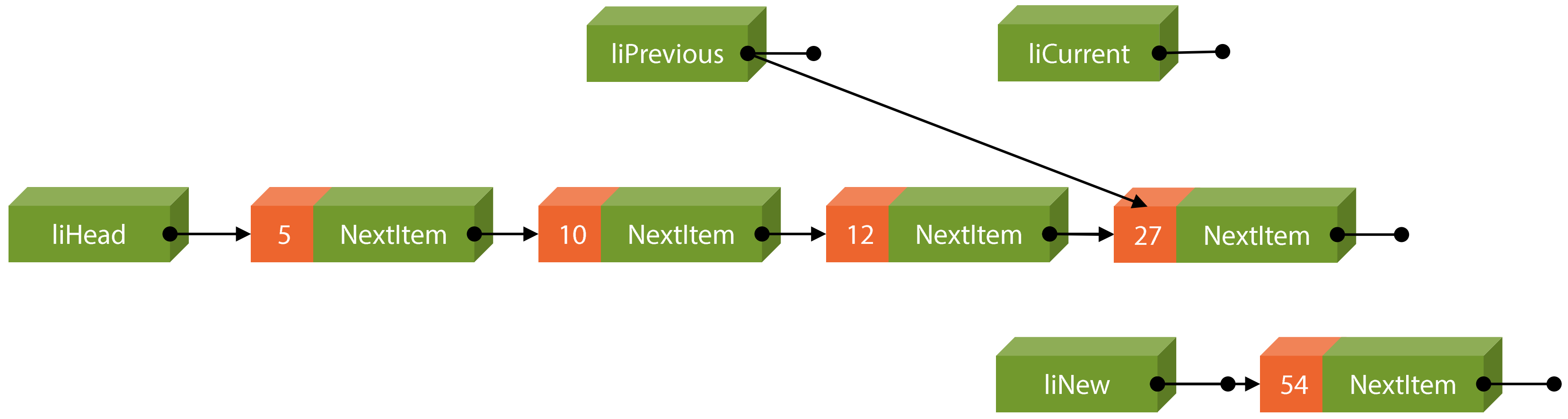
```
If Not liPrevious Is Nothing Then
    Set liNew.NextItem = liPrevious.NextItem
    Set liPrevious.NextItem = liNew
Else
    Set liNew.NextItem = liHead
    Set liHead = liNew
End If
```

Adding An Item to an Ordered List (Beginning)



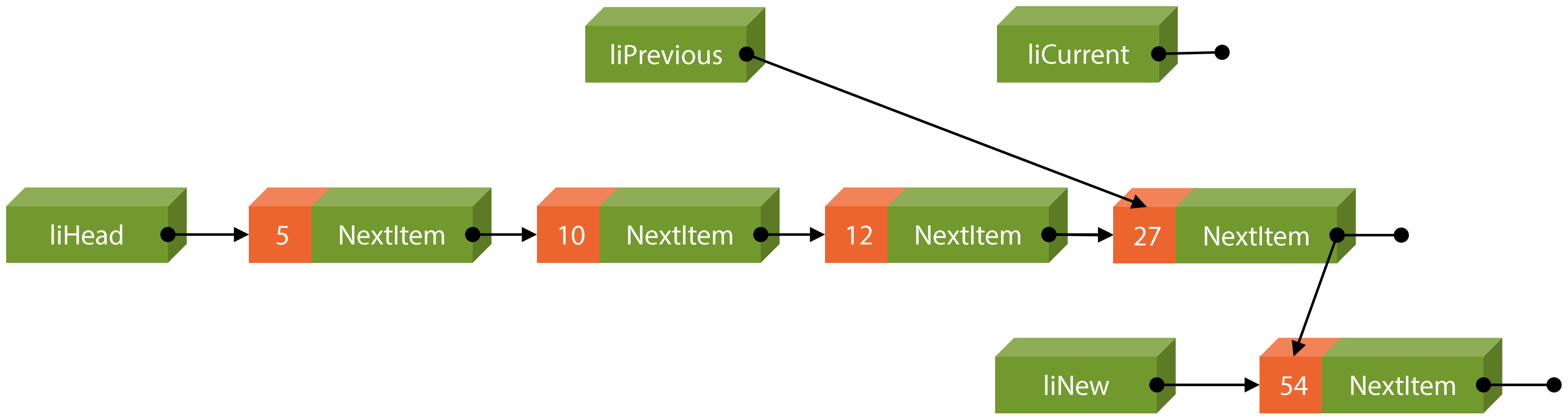
```
If Not liPrevious Is Nothing Then
    Set liNew.NextItem = liPrevious.NextItem
    Set liPrevious.NextItem = liNew
Else
    Set liNew.NextItem = liHead
    Set liHead = liNew
End If
```

Adding An Item to an Ordered List (End)



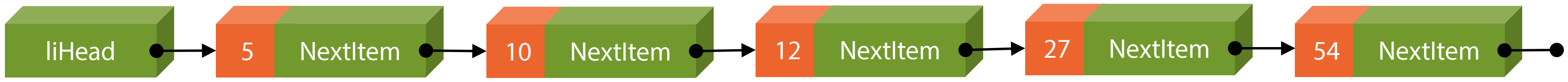
```
' varValue = 54
Set liNew = New ListItem
liNew.Value = varValue

Call Search(varValue, liCurrent, liPrevious)
```

Adding An Item to an Ordered List (End)



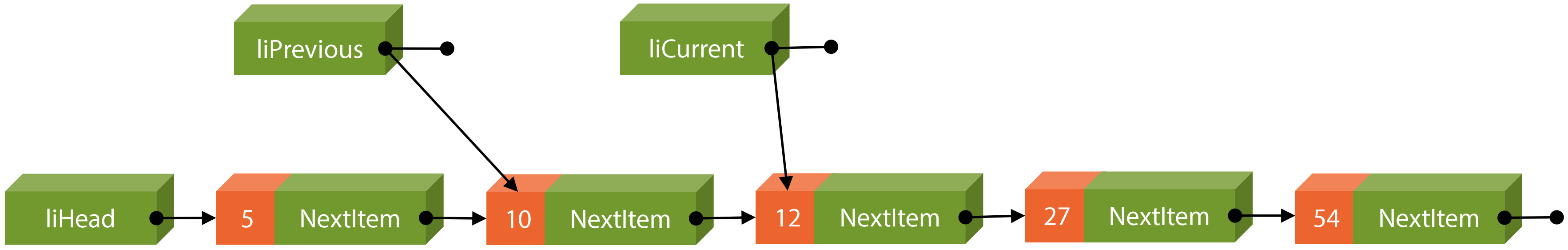
```
If Not liPrevious Is Nothing Then
    Set liNew.NextItem = liPrevious.NextItem
    Set liPrevious.NextItem = liNew
Else
    Set liNew.NextItem = liHead
    Set liHead = liNew
End If
```

Adding An Item to an Ordered List (End)



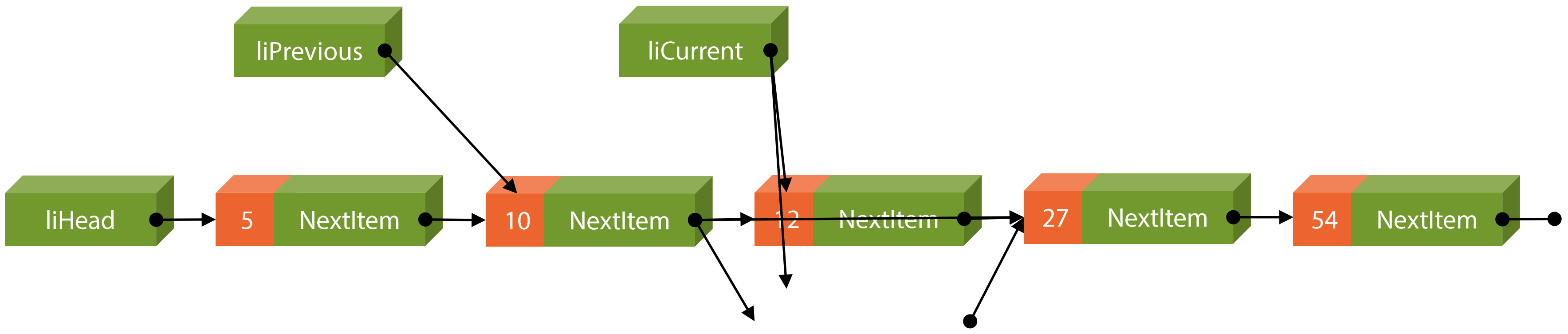
```
If Not liPrevious Is Nothing Then
    Set liNew.NextItem = liPrevious.NextItem
    Set liPrevious.NextItem = liNew
Else
    Set liNew.NextItem = liHead
    Set liHead = liNew
End If
```

Deleting an Item (Middle)



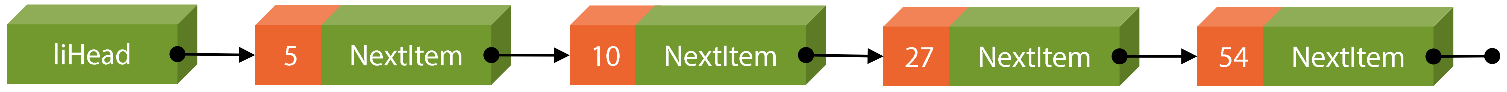
```
' varValue = 12
blnFound = Search(varValue, liCurrent, liPrevious)
' If blnFound = False, nothing else to do!
```

Deleting an Item (Middle)



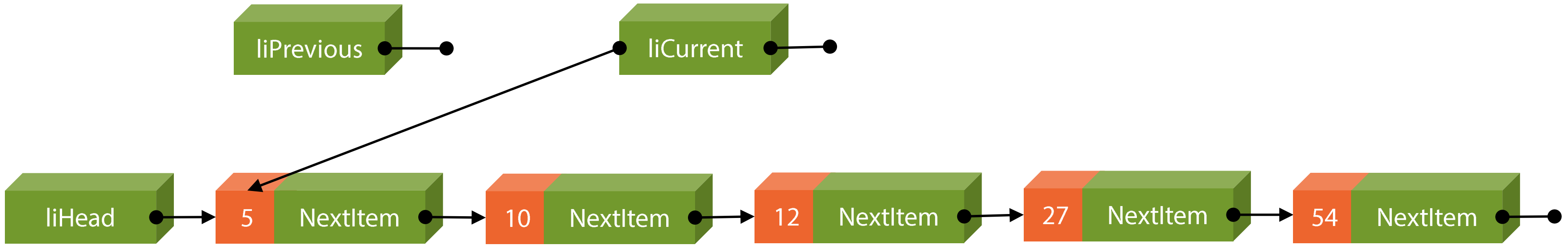
```
If liPrevious Is Nothing Then
    ' Deleting from the head of the list
    Set liHead = liHead.NextItem
Else
    ' Deleting from the middle or end of the list
    Set liPrevious.NextItem = liCurrent.NextItem
End If
```

Deleting an Item (Middle)



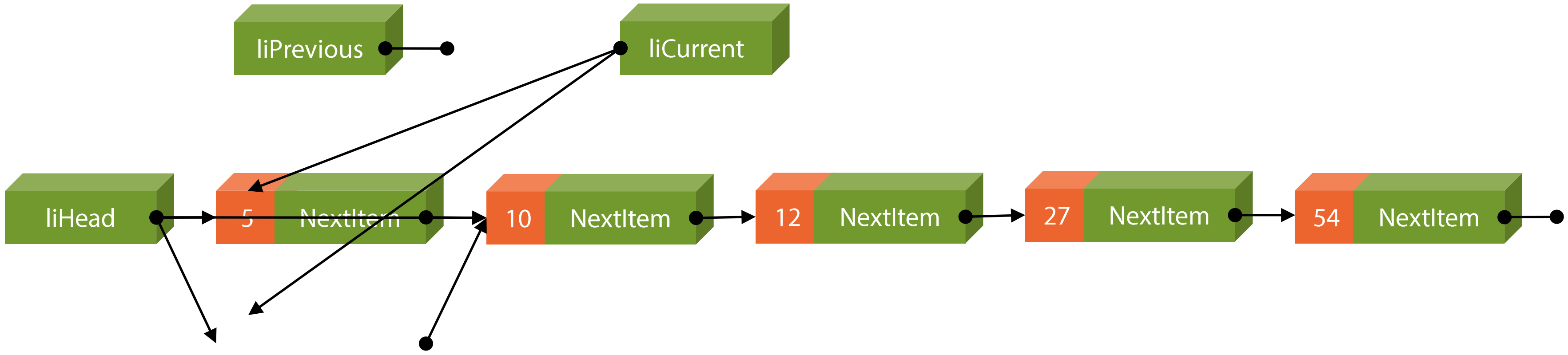
```
If liPrevious Is Nothing Then
    ' Deleting from the head of the list
    Set liHead = liHead.NextItem
Else
    ' Deleting from the middle or end of the list
    Set liPrevious.NextItem = liCurrent.NextItem
End If
```

Deleting an Item (Head)



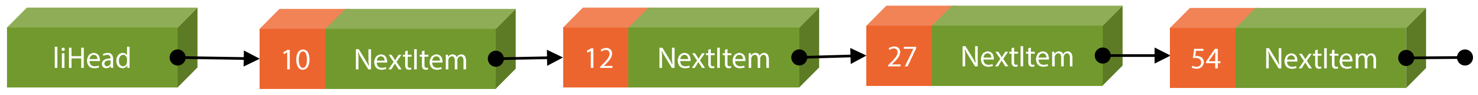
```
' varValue = 5
blnFound = Search(varValue, liCurrent, liPrevious)
' If blnFound = False, nothing else to do!
```

Deleting an Item (Head)



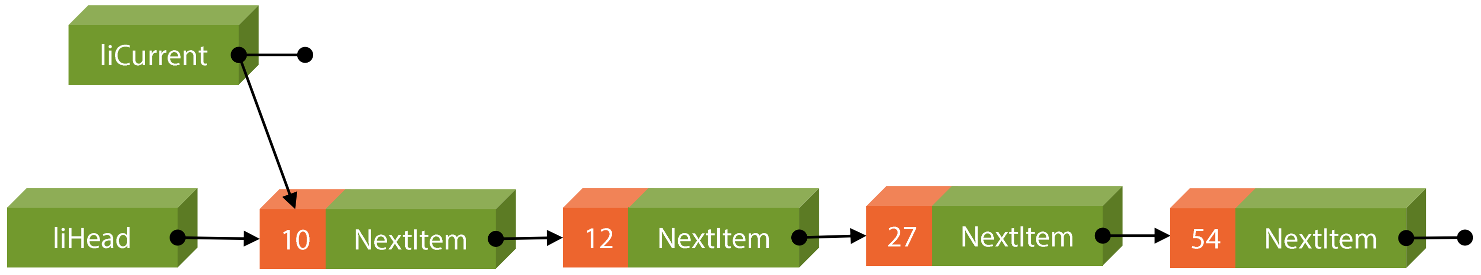
```
If liPrevious Is Nothing Then
    ' Deleting from the head of the list
    Set liHead = liHead.NextItem
Else
    ' Deleting from the middle or end of the list
    Set liPrevious.NextItem = liCurrent.NextItem
End If
```

Deleting an Item (Head)



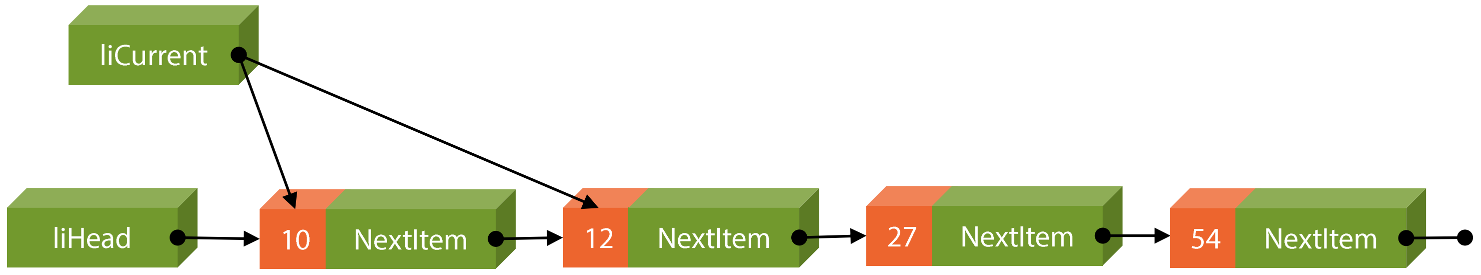
```
If liPrevious Is Nothing Then
    ' Deleting from the head of the list
    Set liHead = liHead.NextItem
Else
    ' Deleting from the middle or end of the list
    Set liPrevious.NextItem = liCurrent.NextItem
End If
```

Traversing a List



```
Dim liCurrent As ListItem
Set liCurrent = liHead

Do Until liCurrent Is Nothing
  ' Work through list items
Loop
```

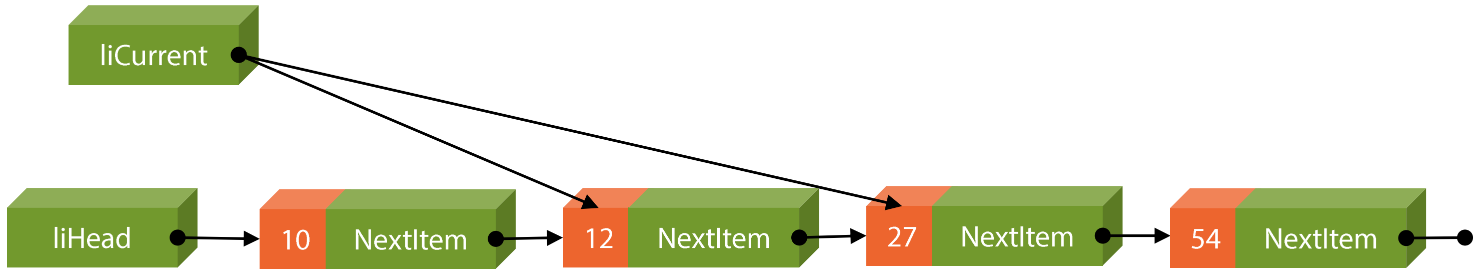
Traversing a List



```
' In the loop, waiting for
' liCurrent to be Nothing
Debug.Print liCurrent.Value
Set liCurrent = liCurrent.NextItem
```

```
10
```

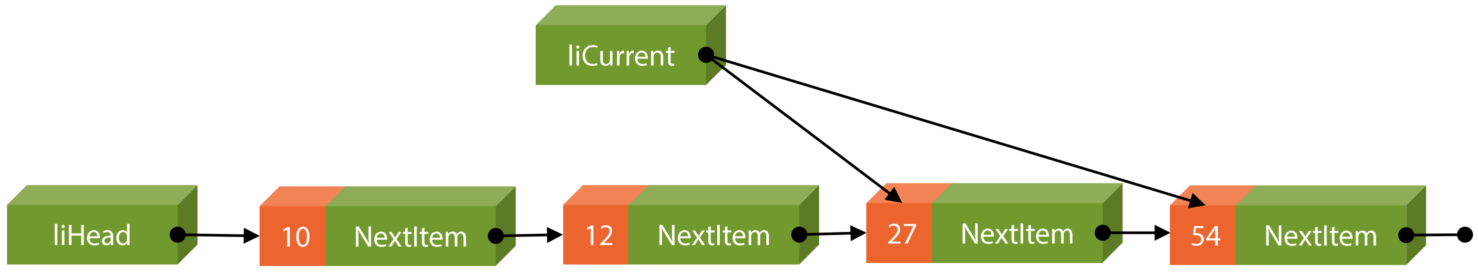
Traversing a List



```
' In the loop, waiting for
' liCurrent to be Nothing
Debug.Print liCurrent.Value
Set liCurrent = liCurrent.NextItem
```

```
10
12
```

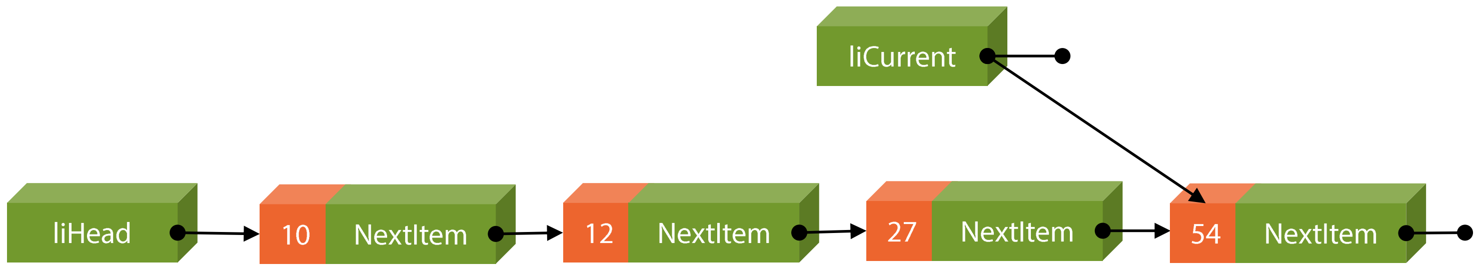
Traversing a List



```
' In the loop, waiting for
' liCurrent to be Nothing
Debug.Print liCurrent.Value
Set liCurrent = liCurrent.NextItem
```

```
10
12
27
```

Traversing a List



```
' In the loop, waiting for
' liCurrent to be Nothing
Debug.Print liCurrent.Value
Set liCurrent = liCurrent.NextItem
' liCurrent is Nothing, so you're done!
```

```
10
12
27
54
```

DEMO

- Run ListTest

Why Use Linked List?

- **Think about it:**
  - VBA's Collection class is similar
  - But not ordered
- **Sorting Collection difficult**
- **Want an ordered list?**
  - Linked list makes it easy

Summary

- **As always, barely covered enough to get started**
- **Much, much more information available**
- **Can easily extend existing knowledge to other data structures**
    - Create one class for structure element
    - Another class for structure header
- **Binary trees, hash tables, and much more**
- **Easy to extend sample classes to add more functionality**